Can bug fix locality and fix date help identify repeated SStuBs?

Mohamed Diallo, Lee Wallace, Omar Beyhum, Constantin Ehrensberger, Lukas Cerny
University College London, UK

***Abstract*—This paper presents a comprehensive study of bug locality within the ManySStuBs4J dataset. By applying the locality sensitive hashing (LSH) algorithm, we identify repeated SStuBs across the dataset and classify these into clone-groups/buckets. Upon these groups we analyzed the time and package locality. From our findings we observe that in 63% of the examined projects, all buckets constrain themselves to a single package. Furthermore, our findings also conclude that time locality is important: for 90% of the projects, more than 80% of repeated SStuBs are patched within the same day. From our results we conclude that package and time-locality are two important factors in bug detection. Our results can go towards improving automated program repair tools and providing developers with the information to prevent these bugs from occurring.**

I. Introduction

Bug identification and fixing forms a road-block for many development projects. Several studies [7] have shown that reworking defects consumes 40-50% of the total cost of software development. Furthermore, these studies [8] have shown that preventing defects can reduce repair time, as prevention is significantly easier than detection.

Clearly, it becomes of interest to see how bugs manifest themselves and how these fixes come to fruition. This knowledge can help automated-program repair tools and developers engage in the prevention of bugs before software is shipped. Ultimately, this will ensure stability of code and lead to lower software-development costs.

One of the issues within automated program repair tools is that they wish to have both high precision and recall. Due to this requirement a good starting-point is focusing on simple easily identified bugs. This brings us to Simple Stupid Bugs (SStuBs).

Simple Stupid Bugs (SStuBs) are bugs that involve a single statement and can be fixed by amending that statement. Although they are often easy to fix, they can be among the most frequent causes of errors in software projects [5]. Many tools such as linters have been developed to detect such bugs, however they do not tend to take into account locality or version history to better detect repeated bugs.

Our study is based on the ManySStuBs4J dataset that contains 153,652 mined SStuBs and their fixes from 1,000 popular large open-source projects [6]. From that we filter it down to 29,057 SStuBs from 410 projects that are distributed under the Apache License. On this dataset we analyse the characteristics of *repeated* SStuBs, those are SStuBs having similar or even identical code, but occurring in different places of the source code. We call a set of repeated SStuBs a *clone group* or a *bucket*. This study analyses the locality and time of fixes for these clone groups. In particular, we want to answer the following research questions:

RQ1: Are repeated SStuBs located within the same package?

RQ2: Are repeated SStuBs fixed within a certain time interval?

If we find that a significant number of repeated SStuBs are fixed within a certain time-frame and only span code within the same package (or module), such results is of great value. Firstly, it is valuable insight for future and current tools that automatically detect bugs as it vastly narrows the search space, enabling them to find bugs more efficiently and reliably. Secondly, it gives developers a better understanding on where to look for related bugs when fixing one, and an insight on the estimated lifespan of repeated bugs within the project.

Our primary contribution within this paper is a comprehensive locality analysis of package and time-intervals containing repeated SStuBs across 410 software projects. The tools and codebase used can be found at **github.com/Diallo/Repated-SStuBs-Analysis**.

II. Background

Several studies have observed that bugs usually manifest themselves multiple times within different locations in the source code. Usually this implies that they need the same *repeated* fix. Yue et al. empirically analysed 3 large open-source software projects (Eclipse JDT, Mozilla Firefox and Libre Office) and found that 15-20% of all bugs needed repeated fixes. Furthermore, when dealing with repeated bug fixes authors found locality to be particularly relevant as between 86% and 99% of repeated fixes were applied within the same package [3]. Such fixes often deal with code clones which all contain the same bug but have not been updated at the same time.

Exploiting bug locality to predict errors is a tried and tested approach that has led to consistent improvements in bug detection [2]. "Clever" [4], an SCM with clone-detection features, tries to address this problem by finding code clones which could cause repeated bugs if only a subset of the clones is modified. However, it forces teams to lose out on the benefits of standardized SCMs. A more lightweight solution could be devised if we determine that repeated bug fixes are most likely

to involve partially updated code clones within a certain time-frame or package.

By studying the locality and fix dates of similar bug fixes within the ManySStuBs4j dataset, we could verify if Yue et al.'s findings hold for a much larger dataset (410 projects as opposed to 3) and learn if repeated fixes are bounded by time (an aspect which has nott been considered in previous work).

III. Experimental Setup

In this section we detail the two-step methodology towards conducting our study. We explain how we prepared the dataset and how we classified code clones using the locality sensitive hashing (LSH) algorithm.

A. Data Preparation

The ManySStuBs4J paper provides a dataset with 153,652 single-statement bug fixes from 1,000 open-source Java projects [6]. This dataset was filtered further to only include 29,057 SStuBs from 410 projects distributed under the Apache License. Then, we removed all duplicated SStuBs, and SStuBs composed of only digits, to get to 28,715 SStuBs from 410 projects. Finally, we augmented our data with commit timestamps by scraping the GitHub API.

B. Locality Sensitive Hashing

Locality Sensitive Hashing (LSH) is a method for *hashing* similar input vectors to the same *bucket*. Within the context of our research the input vectors are SStuBs, whereas the output buckets are considered as *code-clone groups*. Our choice for this algorithm stems from our observations within the dataset: two codes-clones may not always equal an identical match, even though we would like to classify them as belonging to the same group. By applying LSH, we are able to still identify slight changes within repeated SStuBs.

Within the prepared dataset we observed 28,715 SStuBs across 410 projects. We applied LSH to group these into 19,234 buckets. Out of these, only 3,464 buckets contained more than 1 element, thus this is an indication of repeated SStuBs. This provided us with 12,945 SStuBs within 3,464 buckets, which already shows that 15,770 (54.92%) SStuBs were identified as non-repeated SStuBs. However, 12,945 (45.08%) SStuBs were repeated. This confirms our first hypothesis and corresponds to the results from background studies: a large portion of SStuBs tend to be repeated SStuBs.

IV. Results

A. Analysis of Clone Group Size distribution

Of the 3,465 buckets containing more than 1 SStuB, we analyzed the clone group distribution in terms of size. Therefore, we grouped the aforementioned clone groups by the number of SStuBs they contain (i.e. Clone Group Size). As seen in Figure 1, the overwhelming majority (97%) of SStuB clone groups contain less than 10 bugs. Most SStuB clone groups contain either 2, 3, or 4 bugs (which account for approximately 70%, 12%, and 8% of all clone groups respectively). This can be related to results of Yue et al. that found 48%-70% of repeated-fix groups contained only 2 fix instances.

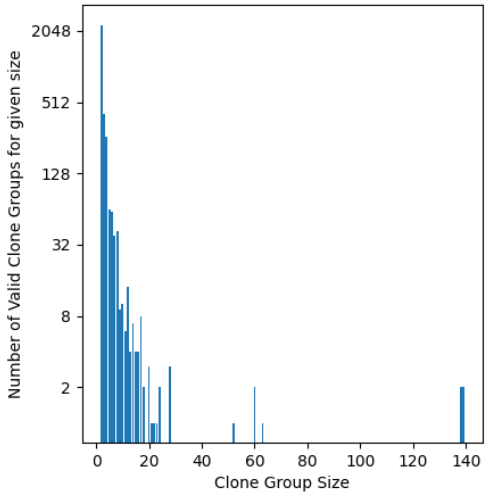

Fig. 1. Distribution of SStuB Clone Group Sizes

Finding: *97% of SStuB clone groups contain less than 10 bugs, with over 70% containing 2 bugs.*

B. RQ1: Are repeated SStuBs located within the same package?

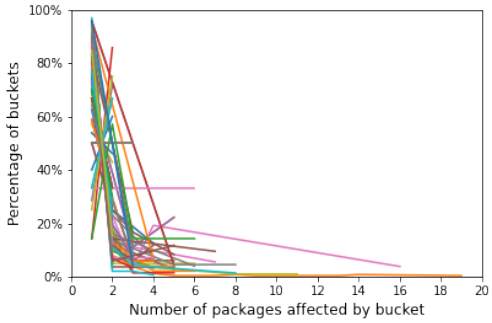

Fig. 2. Affected packages per group of repeated SStuBS for each project

A primary topic of our research, **RQ1**, questions if repeated SStuBs are repeated within the same package. With each line in Figure 2 representing a project, it can be identified that across all projects the large majority of buckets is contained to only a single or up to five packages. Worth noting is that many projects could not be visualized here, as they are composed of only one coordinate and hence cannot be shown as a curve. This is particularly the case for a large number of projects in which all buckets are constrained to a single package, as will be elaborated on later in Figure 4.

It can be observed that there is an overall clear trend from the top left of the chart steep downward, with a few outliers. We observed two types of outliers: those that have a relatively low percentage of buckets allocated to only one package on the center to bottom left, and those few that have a low but nevertheless existent share of buckets that affect a lot of packages each, with the biggest outliers reaching 16 or even 19 affected packages per bucket of repeated SStuBs.

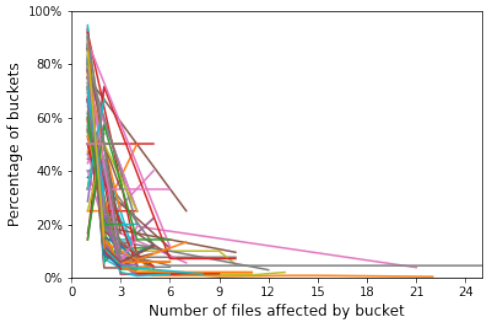

Fig. 3. Affected files per group of repeated SStuBS for each project

Interestingly, a similar pattern of distribution is shown across projects when the percentage of *buckets per project* is measured against the number of files affected by the bucket, instead of the number of packages, as seen in Figure 3. Across the examined projects in the ManySStuBs4J dataset, it can be identified that the general distribution demonstrates that rarely more than 5 files are affected by a bucket.

While this analysis and visualisation provides a general understanding, our further analysis aims at providing a deeper insight into the apparent large share of projects that had most buckets fully contained in a single package or even a single file.

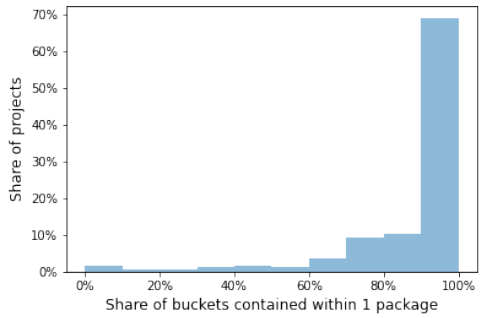

Fig. 4. Distribution of repeated SStuBs limited to a single package

Figures 4 and 5 show histograms that visualise the distribution of projects where all buckets are contained within a single package or a single file.

For the majority of projects, all instances of repeated SStuBs are contained within a single package, and often they are even contained within a single file. For 63% of all projects, 100% of buckets affect only a single package. Furthermore, for 46% of

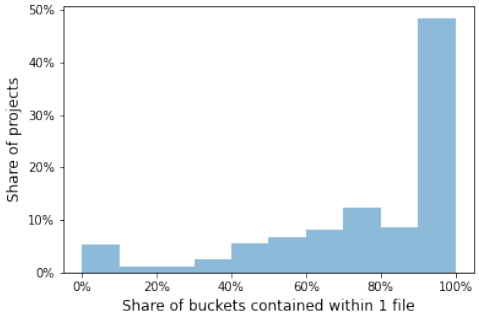

Fig. 5. Distribution of repeated SStuBs limited to a single file

all projects, 100% of buckets affect only one single file. This means that in the majority of cases where repeated SStuBs are contained in a single package, they are also contained in a single file. However, in approximately 50% of the projects we identified that there is at least one category of repeated SStuBs which spread across more than one file.

Our further analysis looked at the trends of repeated SStuBs not contained within a single file. For these SStuBs within multiple files, we found that the majority (70%) would then be located in just two files. Therefore, only 30% of repeated SStuBs are spread across more than two files. Wishing to see the extent of this extreme further, we observed that when we reach the threshold SStuBs split across more than 7 files, we obtain only 5% of the projects.

It is to note that the share of projects where all buckets are contained in a single package appears to be over-proportionately high for projects that have less than 20 different buckets of SStuBs. Since examining this potential relationship exceeds the scope of this paper, it will not be discussed further at this point. Regardless of this observation, as demonstrated in Figure 4, only around 10% of projects appear to have less than 70% of buckets contained in a single package, and for projects with high variation this percentage is even lower. Hence, there is a clear tendency for repeated SStuBs to be mostly contained in a single package.

> **Finding**: *Repeated SStuBs are rarely spread across more than five packages. In over 63% of the examined projects, each bucket of repeated SStuBs is contained to a single package, and in 46% of the projects even to a single file.*

C. RQ2: Are repeated SStuBs fixed within a certain time interval?

Our secondary topic of research, **RQ2** attempts to discover if repeated SStuBs are fixed within a certain time interval. This analysis can identify potential issues surrounding the discovery or the difficulty in patching repeated SStuBs; a valuable insight for those intending to reduce the search space of current or future bug detection tools. repeated SStuBs which received

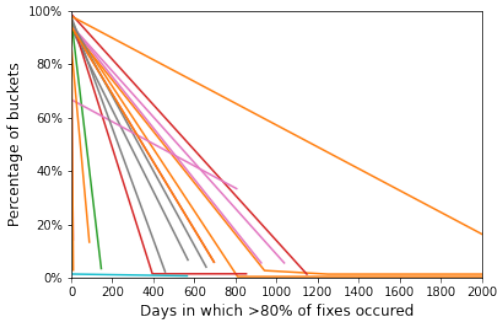

Fig. 6. Fixing time interval per group of repeated SStuBs for each project

same day fixes were categorised as buckets that contained a computed time interval of 0 days. Importantly, to avoid a distortion of the results because of individual outliers in the buckets that were fixed much earlier or later than the majority of SStuBs, this interval is not the time difference between the earliest and latest fix - instead, it is the shortest time interval in which at least more than 80% of fixes were committed.

It is apparent in Figure 6 that most of displayed projects have the vast majority of repeated SStuB fixes occurring on the same day. While this does not mean that they were necessarily fixed in the same commit, the high number of fixes of repeated SStuBs on the same day indicates that teams are typically aware of other instances of a SStuB when fixing one instance.

Further investigation revealed that over 90% of projects have more than 80% of all repeated SStuB fixes occurring on the same day. Positively, it can be inferred that most of the time developers accurately patch a large proportion of repeated SStuBs at the same time. However, as Figure 6 illustrates, when repeated SStuBs are not fixed within the same day, it can take a exceedingly long time for these to be patched. In this dataset, this ranges from 1 day to at times over a year, and in extreme cases even beyond 2000 days (5 ½ years). Thus, demonstrating the need for improvement in source code analysis tools to discover potential similar bugs.

Finding: *For 90% of the projects, more than 80% of repeated SStuBs are patched within the same day. But when this is not the case, it can be observed that the discovery and repair of repeated SStuBs often stretches across a very long time, on average more than a year.*

V. Threats to Validity

Construct: Although LSH allows us to easily surface clone groups without having to pre-define the number or size of groups, it can generate false negatives. Multiple iterations of LSH can be run to reduce the likelihood of false positives. In our methodology, we only ran one iteration of LSH to determine our clone groups due to time and resource constraints. It might be possible that we would encounter less false positives by running multiple iterations of LSH and then merging groups of different iterations that have the same SStuBs.

Projects with a wider variation of SStuBs appear to be less likely to have all repeated SStuBs contained in single packages than projects with less SStuBs variation. Of projects that contain more than 20 different buckets, only 26% have all buckets contained in single packages, and with around 70% the vast majority of projects has between 70% - 90% of buckets contained to a single project. This contrasts with the previously shown results which also include projects with less than 20 buckets, where 69% of projects have over 90% of buckets affecting a single project, and only around 20% of projects have 70% - 90% of buckets affecting a single project. However, independent of SStuBs variation in a project, the vast majority of projects have at least 70% of SStuBs contained in a single package, hence a general tendency remains irrespective of SStuBs variation. While a detailed analysis regarding the package distribution depending on number of buckets per project exceeds the scope of this paper, it should be remarked that this appears to be a factor.

Our final observation is from our finding in Section IV-A: we identify that for 46% of all projects, all the buckets constrain themselves to having only SStuBs within a single file. As our dataset constrains itself to Java projects, we cannot say this will always generalize to practices within other programming languages.

VI. Conclusions and Future Work

This paper presents a comprehensive analysis of SStuBs locality. Our first major finding shows that repeated SStuBs are typically found within one to five different packages. We therefore observed that **RQ1** does hold: the majority of repeated SStuBs found are contained within a single package.

Our second major finding goes to show that time-locality for fixes is important. This brings us to answering **RQ2**: We found that for 90% of the projects, 80% of repeated SStuBs are patched within the same day. For exceptions to this finding, these are not resolved within a timely manner. On average these exceptions take over one year to be fixed. We likely attribute this to the likelihood of simple-bugs being unnoticed as the code-base grows over time.

Our work has shown the importance of considering the locality of bug-fixes. The analysis conducted in the paper can be used to improve a large portion of automated detection and repair tools. Predominantly, we have shown that repeated SStuBs are usually colocated in the same file, and fixed within the same day. These insights can be used to investigate how reductions in the search space of repair tools can increase their performance as well as reduce false positives.

REFERENCES

[1] S. Clark, S. Frei, M. Blaze and J. Smith, "Familiarity breeds contempt", Proceedings of the 26th Annual Computer Security Applications Conference on - ACSAC '10, 2010.

[2] C. Yang, I. Chen, and C. Fan-Chiang. Exploiting module locality to improve software fault prediction. In *2011 IEEE Ninth International Symposium on Parallel and Distributed Processing with Applications Workshops*, pages 342-347, 2011.

[3] R. Yue, N. Meng, and Q. Wang. A characterization study of repeated bug fixes. in *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 422-432, 2017

[4] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, and T. N. Nguyen. Clone-aware configuration management. In *2009 IEEE/ACM International Conference on Automated Software Engineering*, pages 123-134, 2009.

[5] H. Osman, M. Lungu, and O. Nierstrasz. Mining frequent bug-fix code changes. In *2014 Software Evolution Week - iEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 343-347, 2014.

[6] R. Karampatsis, C. Sutton. How Often Do Single-Statement Bugs Occur? The Many SStuBs4J Dataset. In *Proceedings of the 17th International Conference on Mining Software Repositories*, pages 573-577, 2020.

[7] Boehm, Barry W. "Improving software productivity." in *IEEE Computer*, pages 43-57, 1987.

[8] Jones, Capers. Assessment and control of software risks. in *Yourdon Press*, 1994.

VII. Ethical Considerations

Within our project we utilize mined repository data from the ManySStuBs4J dataset. This dataset has been further filtered to constrain itself only to those projects whose license allows for the kind of analysis we perform. Furthermore, we augmented this dataset slightly by adding solely timestamps of the corresponding commit by scraping the GitHub API.

Although we did not explicitly ask each individual repository owner for permission (informed consent), we do believe this type of analysis to be within the context of allowed-usage.

Furthermore, of the information within our dataset we do not use any personable identifiable information, as the developer that authored a specific SStuBs is of no interest to our research.